```cpp
#include <Windows.h>
#include "Level.h"
#include <iostream>
#include <fstream>
#include "Player.h"
#include "Enemy.h" // derived Placeable Actor Classes
#include "Key.h"
#include "Door.h"
#include "Goal.h"
#include "Money.h"
#include <assert.h>

using namespace std;

Level::Level()
    : plevel(nullptr)
    , height(0)
    , width(0)
{

};

Level::~Level()
{
    if (plevel != nullptr)
    {
        delete[] plevel;
        plevel = nullptr;
    }

    while (!m_pActors.empty())
    {
        delete m_pActors.back(); // return us the elements at end, then delete
        m_pActors.pop_back(); // continue to delete the remaining vector elements.
    }
};

bool Level::LoadLevel(string levelName, int* playerX, int* playerY)
{
    levelName.insert(0, "../");
    ifstream levelFile;
    levelFile.open(levelName);
    if (!levelFile)
    {
        cout << "An error has occured." << endl;
        return false;
    }
    else
    {
        constexpr int tempSize = 25;
        char temp[tempSize];
```

```
52
53          levelFile.getline(temp, tempSize, '\n');
54          width = atoi(temp); // converts integer into width.
55
56          levelFile.getline(temp, tempSize, '\n'); // line 83 and line 87 link.
57          height = atoi(temp);
58
59          plevel = new char[width * height]; // array that we need to deallocate.
60          levelFile.read(plevel, width * height);
61
62          if (playerX != nullptr && playerY != nullptr)
63          {
64              bool anyWarnings = Convert(playerX, playerY);
65              if (anyWarnings)
66              {
67                  cout << "There are some warnings in the level data. see above." << endl;
68                  system("pause");
69              }
70          }
71          return true;
72      }
73  }
74
75  void Level::Draw()
76  {
77      HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE); // temprary variables being deleted at the end of draw.
78      SetConsoleTextAttribute(console, (int)ActorColour::Regular);
79
80      //Draw Level
81      for (int y = 0; y < GetHeight(); ++y)
82      {
83          for (int x = 0; x < GetWidth(); ++x)
84          {
85              int indexToPoint = GetIndex(x, y);
86              cout << plevel[indexToPoint];
87          }
88          cout << endl;
89      }
90
91      COORD actorCursorPosition; // position the cursor at correct location, x and y variables
92
93      // Draw actors
94
95      for (auto actor = m_pActors.begin(); actor != m_pActors.end(); ++actor) // going to the beginning and through the end.
96      {
97          if ((*actor)->IsActive()) // if active we want to draw.
98          {
 99             actorCursorPosition.X = (*actor)->GetXPosition();
100             actorCursorPosition.Y = (*actor)->GetYPosition();
101             SetConsoleCursorPosition(console, actorCursorPosition); // set position manually to this point.
102             (*actor)->Draw(); // draw the actors, tempoary variable in line 93 is now finished and deleted from the stack.
103         }
104     }
105 }
106 
107 bool Level::IsSpace(int x, int y)
108 {
109     return plevel[GetIndex(x, y)] == ' ';
110 }
111 bool Level::IsWall(int x, int y)
112 {
113     return plevel[GetIndex(x, y)] == WAL;
114 }
115 
116 bool Level::Convert(int* playerX, int* playerY)
117 {
118     bool anyWarnings = false;
119 
120     for (int y = 0; y < height; ++y)
121     {
122         for (int x = 0; x < width; ++x)
123         {
124             int intIndex = GetIndex(x, y);
125 
126             switch (plevel[intIndex])
127             {
128             case '+':
129             case '-':
130             case '|':
131             {
132                 plevel[intIndex] = WAL;
133                 break;
134             }
135             case ' ':
136             {
137                 break;
138             };
139             case 'r':
140                 plevel[intIndex] = ' ';
141                 m_pActors.push_back(new Key(x, y, ActorColour::Red));
142                 break;
143             case 'g':
144                 plevel[intIndex] = ' ';
145                 m_pActors.push_back(new Key(x, y, ActorColour::Green));
146                 break;
147             case 'b':
148                 plevel[intIndex] = ' ';
149                 m_pActors.push_back(new Key(x, y, ActorColour::Blue));
```

```cpp
                break;
            case 'R':
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Door(x, y, ActorColour::Red, ActorColour::RedSolid));
                break;
            case 'G':
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Door(x, y, ActorColour::Green, ActorColour::GreenSolid));
                break;
            case 'B':
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Door(x, y, ActorColour::Blue, ActorColour::BlueSolid));
                break;
            case 'X':
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Goal(x, y));
                break;
            case '$':
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Money(x, y, 1 + rand() % 5));
                break;
            case '@':
            {
                plevel[intIndex] = ' ';
                if (playerX != nullptr && playerY != nullptr)
                {
                    *playerX = x;
                    *playerY = y;
                }
                break;
            }
            case 'e':
                m_pActors.push_back(new Enemy(x, y));
                plevel[intIndex] = ' '; // clear level
                break;
            case 'h': // horiztonal enemy
                m_pActors.push_back(new Enemy(x, y, 3, 0));
                plevel[intIndex] = ' ';
                break;
            case 'v': // vertical enemy
                plevel[intIndex] = ' ';
                m_pActors.push_back(new Enemy(x, y, 0, 2));
                plevel[intIndex] = ' ';
                break;
            default:
            {
                cout << "Invalid character in file " << plevel[intIndex] << endl;
                anyWarnings = true;
                break;
            }
            }
        }
    }
    return anyWarnings;
}

int Level::GetIndex(int x, int y)
{
    return x + y * width;
}

// Updates all actors and returns a colliding actor is there is one.

PlaceableActor* Level::UpdateActors(int x, int y ) // pass in x and y of player.
{

    PlaceableActor* collidedActor = nullptr;

    for (auto actor = m_pActors.begin(); actor != m_pActors.end(); ++actor)
    {
        (*actor)->Update(); //update all actors

        if (x == (*actor)->GetXPosition() && y == (*actor)->GetYPosition()) // collision occured
        {
            assert(collidedActor == nullptr); // if assertion fails, two points have met.
            collidedActor = (*actor); // points to the location of the collision.
        }
    }
    return collidedActor;
}
```